A Performance Study of Method Execution Alternatives in a Distributed Object Database System

Shalab Goel

Bharat Bhargava
Purdue University, bb@cs.purdue.edu

Report Number:
95-037

A PERFORMANCE STUDY OF METHOD EXECUTION ALTERNATIVES IN A DISTRIBUTED OBJECT DATABASE SYSTEM

Shalab Goel
Bharat Bhargava

Department of Computer Science
Purdue University
West Lafayette, IN 49707

CSD-TR-95-037
June 1995

A Performance Study of Method Execution Alternatives in a Distributed Object Database System*

Shalab Goel Bharat Bhargava
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

Efficiency in query processing is a major issue to achieve performance in future database applications. Dynamic method execution and object referencing (both inter and intra object referencing) are two features of object query languages like SQL-2 and SQL-3. These features add great power to the users in presenting their inqueries to the database systems. In this paper we describe the model and implementation of dynamic method (or user-defined procedures) execution during query processing, as supported by the O-Raid distributed object-oriented database system. The methods invoked on objects include constructor methods for insert queries during initializing of the objects and predicate methods for selection queries. To achieve greater extensibility to an object-oriented system, our model allows user-definable methods to be tailored as users desire, and to be loaded on demand. It also supports object fragmentation and replication in distributed object-oriented systems. Data as well as executable method codes can be fully or selectively replicated among all sites. In addition to remote data access, we also provide remote method invocation, in which methods are executed on data located on a remote site, and the results are sent back. This paper also presents the results of our experimental studies. We use the O-Raid distributed database system in our experiments. We demonstrate how the query processing functionality can be improved using user-defined C++ programs, and how new functions such as the pattern matching can be added to the database system. We develop a benchmark for dynamic method execution in queries and perform measurements. The overheads in dynamic execution of user-defined procedures in queries are identified and analyzed. We also study the overheads for data migration (getting data from remote site and processing them locally) versus method migration (processing query on remote site and bring back the results) in distributed query processing. We discuss the heuristics on how to choose the schemes to improve efficiency of distributed query processing.

*This research is supported in part by a grant from AIRMICS and UNISYS.

1 Introduction

Efficiency in object query language processing is a major issue in achieving performance in distributed "non-traditional" database applications. The two main features of object query language standards such as SQL-3 and ODMG are the support for method invocations and object referencing in the user queries. These features add great power to the users in presenting their queries to the database systems.

Many proposals have studied the performance optimization of object queries in distributed environments. The object query execution models presented by Kim et. al. in [1], Valduriez et. al. in (cite Rozaquette) , and Bertino in [2] have focused on optimizing the performance of subobject references (or path specifications) in the object queries. A subobject reference is mapped into a set of semi-join and join operations which is optimized for the optimal execution sequence. The objects in distributed query model are assumed to be configured in LAN environment which provides similar communication delays between any two sites of the database. Efficient access mechanisms to support subobject referencing have been proposed in [3, 4]. Thus, most of this work focuses on efficient object traversal in distributed object databases.

For supporting method invocation in object queries, Bertino [5] and Kemper [6] have proposed the method pre-computation approach. In this approach, the method is computed on an object and the results of this computation stored as attributes of the object in the database. These attributes are called *derived* attributes for the object class in [5]. The method invocation in a user query later can be completed by retrieving these attributes from the database. The approach of precomputing the methods, although attractive and efficient, is limited in its applicability due to the following assumptions in the model:

- Only the internal state of the object can be used in method computation. This eliminates the methods that specifies external arguments from getting pre-computed.
- Only the primitive type attributes can participate in the method computation. This means a method defined on a complex object class (an object containing references to other user-defined class objects) and which invokes methods on (or accesses) its subobjects cannot be

precomputed.

These restrictions limit the scope of methods that can be defined on object classes in the database schema. Even if we assume that these limitations do not exist and that all methods invocations in the user queries can be processed by retrieving the stored resullts from the database, the methods still need to be executed either for initial computing of the derived attributes for an object, or for recomputing them when the internal state of the object changes. Thus, the execution of methods in object queries is inevitable if the expressiveness of user queries cannot be compromised. In this paper, we study the performance of two approaches for method execution in object queries on object-oriented databases distributed over different network environments such as local area networks (LAN), metropolitan area networks (MAN), and wide area networks (WAN).

In the first approach, the traditional approach of method execution in an object query, the remote component objects are fetched to the local site (the query site); an in-memory object is assembled locally; and method specified in the query is invoked on this object. This approach of *data migration* may have poor performance when number and size of objects located at remote site are large. The performance may further aggravate in distributed environment such as wide area network which may be both slow and unreliable. The alternative approach is to migrate the method execution to a remote site - an in-memory object is assembled at the remote site; method is executed on the assembled object at the remote site; and the results of method execution are returned to the local site where the query processing is resumed. This approach of *method migration* may minimize the cost of method execution if the component objects are located either locally or in close proximity of this method execution site. However, there are overheads of initiating a remote method execution request and retrieving the remote computed results. We present the experimental evaluation of these approaches in the following scenarios:

1. *The size of object is large:* It may be better to migrate the method execution to the remote site where the large object (for example, multimedia object) is located than to bring it locally over a possibly slow and unreliable communication channel.

2. *The object is complex and distributed over multiple sites:* The complex object may be fragmented on multiple sites to provide better availability and performance to the database

users [7]. It may be better to migrate method execution to a remote site where the cost of composing the complex object will be low.

We present a heuristic that uses the communication delays between any two sites, and the size and the configuration of object on distributed sites to determine the appropriate method execution approach. The heuristic also determines the optimal cost for migrating the method execution to a remote site. We use a distributed object-oriented database system called O-Raid [8] developed at Purdue University, as our testbed for implementing and analyzing the performance of these method execution approaches in various network environments. We have integrated O-Raid with WANCE tool [13], a Wide Area Network Communication Emulator, to provide real wide area network communication environment for conducting our experimental studies.

The rest of the paper is organised as follows. Section 2 presents the overview of O-Raid system and the support for method execution in O-Raid. Section 3.1 describes the heuristic for determining the site with optimal cost of method execution. Section 4 presents the database benchmark and the experimental facilities such as WANCE tool for our experimental study. Section 5 evaluates the method execution schemes and the validity of our heuristic. Section 6 concludes the paper with the directions about our current and future proposed work.

2 Supporting Method Execution in O-Raid

O-Raid supports object-oriented features on top of Raid [9] relational distributed database system. OpenODB [10], UNISQL/X [11] and Oracle Relation-Object product [12] are some of the commercial systems that use a similar layered approach for supporting objects on top of an existing relational system.

2.1 Overview of O-Raid System

The O-Raid system supports a hybrid object-relation data model. The basic unit of storage is still a relation. However, the attributes are not limited to simple data types such as integer and strings, but can be arbitrary complex objects of pre-defined user C++ [14] classes. O-Raid supports both

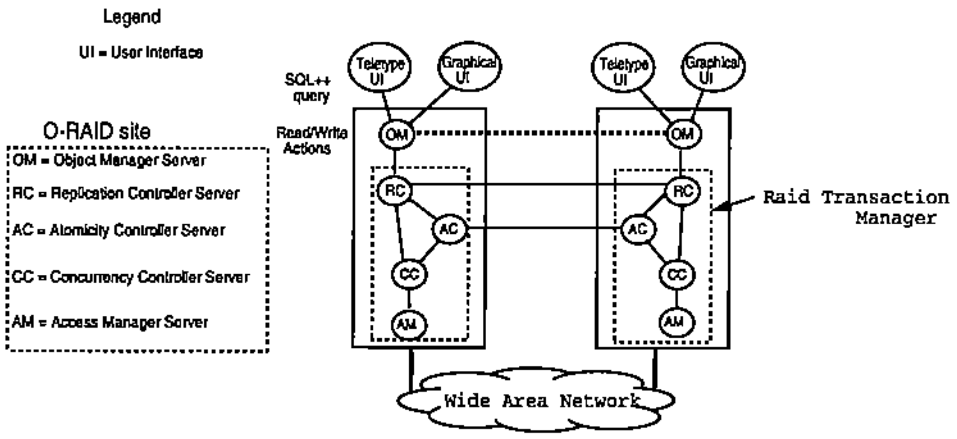
Legend
UI = User Interface
O-RAID site
OM = Object Manager Server
RC = Replication Controller Server
AC = Atomicity Controller Server
CC = Concurrency Controller Server
AM = Access Manager Server
SQL++ query
Read/Write Actions
Teletype UI
Graphical UI
OM
RC
AC
CC
AM
Raid Transaction Manager
Wide Area Network

Figure 1: Architecture of a two site O-Raid system

inter-object referencing (or pointer referencing) and intra-object referencing (or embedded object referencing, where an object is stored with another object).

The O-Raid system site consists of an Object Manager (OM) server and an instance of Raid Transaction Manager (TM) as shown in figure 1. The OM server manages all aspects of object manipulation and generates a transaction consisting of reads and writes operations (actions) which is submitted to Raid TM for further processing. Thus, the underlying Raid servers only deal with transactions consisting of operations on relations, while the OM server provides support for object-oriented features to the object database designers. O-Raid uses SQL++ [15] query language interface for user queries involving objects. Further details about O-Raid can be found in [7].

Object Swizzling and Unswizzling in O-Raid Each class definition A in an O-Raid database schema is associated with a unique *class relation* A stored in the underlying Raid relational database. The objects of a class are stored as the tuples of the corresponding class relation. Each pointer reference "b" of class B in the definition of class A is stored as a 3-tuple called *persistent pointer* in the class relation A as shown in figure 2. This persistent pointer identifies uniquely, for each class A object stored in the class relation A, the tuple of class relation B that corresponds to the referenced class B object.

To compose an in-memory object of class A , the OM server must *recursively* request from Raid

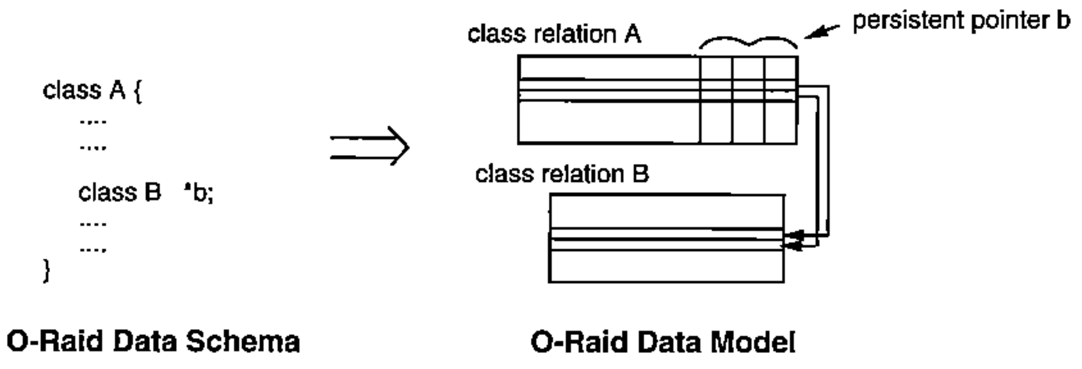

Figure 2: Hybrid Object-Relation Data Model in O-Raid

TM the tuple corresponding to the subobjects referenced in class A object. The persistent pointer in the class relation A tuple uniquely identifies the class relation B tuple. OM server submits a read request to Raid TM to retrieve this tuple. The tuple returned by Raid TM is used to complete the assembly of an in-memory class A object as follows – the retrieved tuple is used to compose an in-memory object of class B [1]; and the pointer reference "b" in the class A object is updated to point to this class B object. Further details about the engineering of object-relation model can be found in [16].

Method Execution in O-Raid O-Raid supports the dynamic execution of methods in SQL++ queries. A dynamic linking library package called *dld* [17], available from GNU Free Software Foundation, is used for dynamically loading and linking the precompiled code for the method invoked in the user query with the OM server process at the run time.

A method invocation in the SQL++ query is processed by the OM server as follows:

- An in-memory object of the base attribute class (for example, a->b.c->d.proc() has base attribute d) is composed at the query site as described above above.
- The object code module for the base attribute class is loaded into the memory, if it is not already there. The method invoked in the query is searched for in object code. A dld library

[1]If class relation B tuple itself contains persistent pointers, they will be swizzled first.

routine returns a pointer to the beginning of method code in the object code. An explicit function call with the arguments specified in the query is executed.

O-Raid supports *data migration* since the objects distributed on multiple sites will be migrated to the query site for method execution. The data migration is transparent to the users and is achieved by the Raid TM's Replication Controller (RC) server.

3 Building Support for Method Migration in O-Raid

The server-based architecture of O-Raid has provided the ease of design and implementation for supporting method migration approach to method execution. The dashed line between the peer OM servers in figure 1 represents the augmented message based control flow in O-Raid for engineering this support. This provides the query site capability to request remote site for method execution.

Message Control Flow: The interaction between the OM servers at query site and target remote site is realized by a synchronous message exchange as follows: The OM server at query site sends a request message to the OM server at remote site. The request message specifies the method identifier which uniquely identifies the method to be invoked, the arguments for the method invocation, and a persistent pointer to object on which method will be executed. The OM server at remote site parses the message and initiates a subtransaction to process the method invocation request. An in-memory object identified by the persistent pointer is composed and the method executed on it. The results of method computation are returned in an acknowledgement message to the query site. The query processing at the query site is resumed after the result of method computation is extracted from this message. It must be noted that eventhough the subtransaction completes successfully, it does not terminate and release its locks [2] on the data items. Th esubtransaction will terminate only when the query transaction terminates.

Synchronous Method Migration: Our design uses a handshake mode of message interaction between peer OM servers for implementing method migration. We have made this deliberate choice

[2]Raid TM uses two phase locking in Concurrency Controller (CC) server

to eliminate the performance gains in method migration approach due to concurrent execution of different parts of the query at query site and target remote site for method invocation respectively. This allows us to make a fair comparison of method migration approach with data migration approach where the entire query is executed at the query site. At the time of writing of this paper, however, we are investigating the potential gains in performance by allowing asynchronous interaction between the OM servers and thus achieving parallel processing of the query at multiple sites. This is briefly outlined in our future work in section 6.

The OM server uses the cost model described next to determine the approach for execution of the method specified in the query. It also determines the site with optimal cost for method migration. The cost of migrating the method execution consists of the cost for initiating a subtransaction for processing method invocation request, cost of composing an in-memory object from possibly distributed or remotely located object, and the cost of executing the method itself.

3.1 Cost Model for Optimal Site Selection

Given a method $M_{ij}(A^1_{ij}, A^2_{ij}, \ldots, A^n_{ij})$ such that M_{ij} is the j*th* method defined on class C_i and A^k_{ij} k=1(1)n are the formal arguments of M_{ij} with class in { C_1, C_2, ..., C_m } where m is number of class definitions in object database schema; we have the following parameters that describe the class and its attributes:

- S_i : Size of class C_i object
- N_i : Number of attributes of user-defined classes in class C_i object.
- L_i : Location of class C_i objects
- $\text{Comm}_{j,k}$: Communication Cost for transferring unit data from site j to site k.
- RM_{ij} : Size of request message for migration of method M_{ij}.
- Q_S : Query Site

The parameter $\text{Comm}_{j,k}$ models the cost of transferring object from from one site to another site using the real communication delays, and impacts the cost of method migration most significantly in wide area network environment.

To simplify our model and also to make it applicable to O-Raid system, we make the following assumptions:

1. The precompiled code for all the methods is replicated on all sites.
2. The objects of a class are located on one particular site, i.e. there is no replication or horizontal fragmentation of the class.
3. The class attributes are single valued.
4. The formal parameters in method definition are simple typed.
5. The I/O costs are neglible.

At present, O-Raid supports processing on main-memory databases. So the paging and clustering issues are not considered. However, our results are applicable under the conditions when the entire working set of application fits in the workstation's bufferpool.

The cost of migration of execution of method M_{ij} to site l from query site Q, written as Cost_{Total} $(\text{M}_{ij}, \text{l})$, consists of three components:

1. $\text{Cost}_I(\text{M}_{ij}, \text{l})$ Cost of initiating a subtransaction at site l for executing method M_{ij}. This involves the cost of sending a request message from site Q from site l and the startup cost for the subtransaction.
2. $\text{Cost}_A(\text{C}_i, \text{l})$ Cost of in-memory assembly of class C_i object at site l.
3. $\text{Cost}_M(\text{M}_{ij}, \text{l})$ Cost of method invocation at site l. This includes the assembly cost for complex type arguments of method M_{ij}, and the method invocation cost.

$$Cost_I(M_{ij}, l) = Comm_{Q,L_i} * RM_{ij} + \alpha$$

$$Cost_A(C_i, l) = Comm_{L_i,l} * S_i + \sum_{k=1}^{N_i} Cost_A(C_k, l)$$

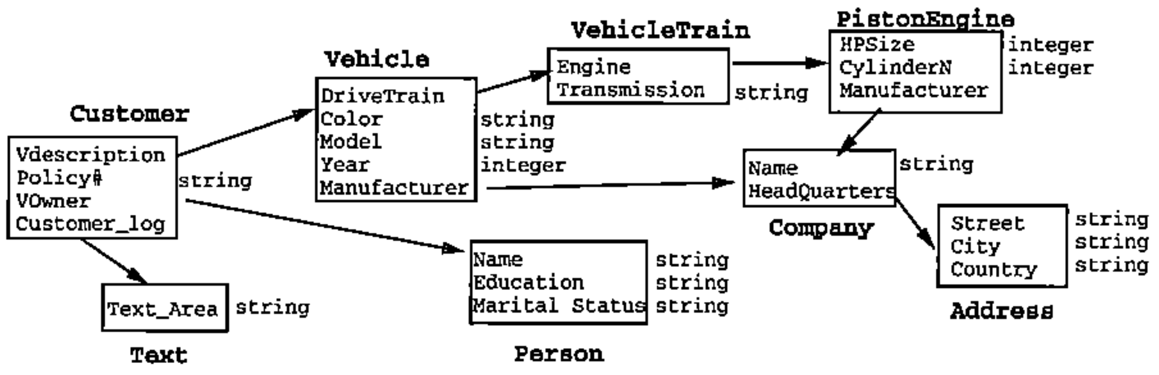

Figure 3: Class Attribute Hierarchy for Customer object

$$Cost_M(M_{ij}, l) = \sum_{k=1}^{n} Cost_A(C_{A_{ij}^k}, l) + Exec(M_{ij}$$

Find site k such that Cost$_{Total}$ (M$_{ij}$, k) is minimized.

4 Experimental Infrastructure

4.1 Benchmark

Database Design Since our main concern is the relative comparison of data migration versus method migration for processing user-defined procedure calls in the object queries under different distribution and fragmentation of composite objects on multiple sites, we use a simple database called Insurance_Agents database. Its basis is a set of *customer* objects. Each customer object specifies the owner of a vehicle and description of the vehicle he/she owns. The information is maintained using an *embedded* object Person and a pointer to object of type Vehicle. The class-attribute hierarchy as defined in [3] and rooted at class Customer is shown in figure 3. The following schema makes the database specifications.

```
class Address {                              class Company {
  char Street[64], City[24];                   char Name[64];
```

```
   char State[12],Country[16];
   Address(char *,char *,char *,char *);
 };


 class VehicleTrain {
   PistonEngine *Engine;
   char  Transmission[24];
   VehicleTrain(char *);
   int Imported_Engine();
 };


 class Vehicle {
   char Model[64], Color[32];
   Company *Manufacturer;
   VehicleTrain *DriveTrain;
   Vehicle(char *,char *);
   int Estimated_Value(int,int);
 };
              };
class Customer {
    Person *VOwner;
    Vehicle *Vdesc;
    char  Policy_No[16];
    class Text Customer_log;
    Customer (char *);
};
```

```
   Address *HeadQuarters;
   int Located_In(char *);
   Company(char *);
 };


 class PistonEngine {
   int HPSize, CylinderN;
   Company *Manufacturer;
   PistonEngine(int,int);
 };


 class Person {
   char Name[64];
   char SS#[16];
   char Education[4];
   char Status[8];
   int EducationLevel(char *);
   Person(char *, char *, char *, char *);


 class Text {
   char **Text_Area;
   int contain_key(char *);
   Text(char *);
 };
```

For each class, we have a constructor method as well as other user-defined methods. The Insurance_Agents database contains the classes as described above and a relation customers that contain a single attribute of type Customer.

Queries and Output Parameters *Select* queries invoking predicate methods on different class objects are defined in the benchmark and are as follows:

- *Query 1:* Find the owners of vehicles with estimated insurance value greater than 50000.
- *Query 2:* Find the owners of vehicles with imported engines.
- *Query 3:* Find the owners of the vehicle manufactured in Midwest USA.
- *Query 4:* Find the name of the vehicle owners with education higher than high school.
- *Query 5:* Find the vehicle owners who have never received a citation for reckless driving.

For each query, we measured the total time for method execution at each site of the database and its component time for assembly of an in-memory object on that site. We also measured the same output parameters for the optimal site selected using our cost model. The timing was obtained by putting probes in the Object Manager server code.

The execution of the same query is repeated to filter out the effects of wide variance in communications performance on the Internet [18] in order to have statistically significant results. The experiments are run at the late night to avoid network load peaks.

4.2 Experimental Facilities

We use Sun SparcStation-1s running SunOS 4.1.1 in the Raid Laboratory as our testbed. All of them have local disks and are connected by a 10 Mb/s Ethernet. Furthermore, they form their own subnet that is separated from departmental network. We use Raid communication subsystem that is an efficient, location-transparent, and transaction-oriented message passing facility [9]. After receiving a user query, the O-Raid system generates transactions for local Raid TM, which will send messages to other Raid TMs when remote access is necessary. The messages are packed into UDP/IP packets and delivered by the networks.

WANCE Tool For wide area network environment, we use a Wide Area Network Communication Emulator (WANCE) tool to emulate Inetrnet communication in a LAN environment [13]. We chose

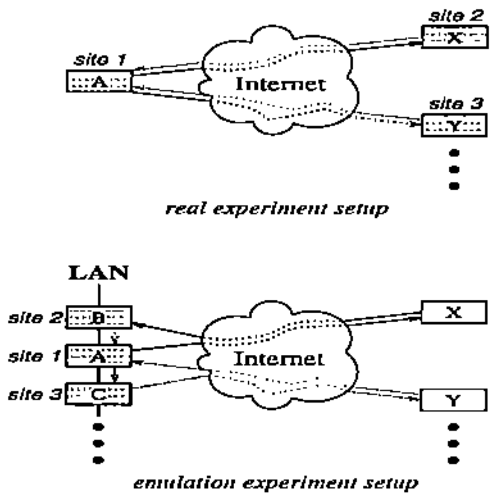

Figure 4: Emulating the real experiment

the emulation approach for our WAN experiments because we can have real WAN communication in a LAN environment. We avoided the simulation experiments because the current simulation models oversimplifies the Internet, which consists of over 1,000,000 hosts around the world. Also, our goal is to evaluate our scheme in the real experiments.

The emulation approach is based on the observation that the difference between the behaviours of a distributed system running on a LAN and that on a WAN is primarily due to the communication performance, not the location of the experimental host. To emulate a two site system spanning site A and site C that spans over WAN, we only need to find another computer B comparable to C but in same LAN as A. We run the O-Raid sites on A and B instead of A and C. The WANCE tool can be specified to automatically route all messages from A to B through C. Although our experiments are run in A and B (two SparcStations in our lab) we have the same effect of running them in A anc C as shown in [18].

5 Evaluation of Method Execution Schemes in O-Raid

5.1 Statement of the Problem

The cost of method invocation on an object in distributed environment will depend on the number and the size of subobjects required for its in-memory assembly. The method migration approach relocates the method execution to remote site (different from query site). Depending upon how the object is configured, this relocation may reduce the cost of object assembly and hence the total cost of method execution. However, as pointed out there are some overheads associated with this approach.

The experiments in this section compare the performance of migrating the method execution to a remote site with the traditional one of migrating the object data to the query site. We compare the cost of object assembly cost as well as total cost of method execution for both the approaches. In the first set of experiments, we compare the performance of these schemes for methods defined on class objects with different number of subobject references. In the second set of experiments, we analyze the impact of the size of the flat object on their relative performance.

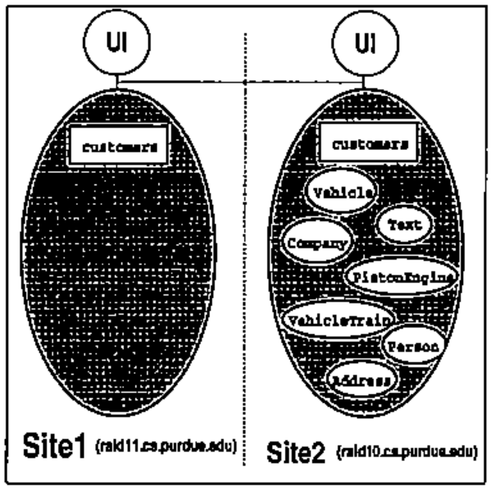

Figure 5: Configuration of Two Site Benchmark Database

5.2 Procedure

Figure 5 shows a two site benchmark database. All the class objects - `Vehicle`, `Text`, `VehicleTrain`, etc. in the benchmark are stored at *site2*. The `customers` relation which contains objects of classes `Vehicle`, `Text`, and `Person` respectively is replicated on both sites.

In our experiments, we consider three alternatives for method execution:

1. **Data Migration** *from* **Local Site** *(DML):* The benchmark queries are submitted at site2. The object on which method specified in the query is to be invoked is composed at *site2*.

2. **Data Migration** *from* **Remote Site** *(DMR):* The benchmark queries are submitted at site1. The object on which method specified in the query is to be invoked is composed at *site1*.

3. **Method Migration** *(MM):* The benchmark queries are submitted at site1; the class object is composed and method invoked on it at *site2*.

We run *site1* and *site2* in `raid11.cs.purdue.edu` (raid11) and `raid10.cs.purdue.edu` (raid10) respectively in our LAN environment. We set up WANCE tool so that raid10 simulates the following Internet sites: `attospm.physics.purdue.edu`, `tajmahal.ecn.purdue.edu`, `teresa.cs.uiuc.edu`, `pegasus.cs.pitt.edu` and `seoul.usc.edu`. The experiments were run at the late night to avoid network load peaks.

Each query is reapeated 50 times to filter out all the factors other than those of O-Raid system in order to have statistically significant results. Furthermore, the executions of the queries are interleaved one by one rather than group by group to allow them to run under similar environment and result in fair comparison.

In the first experiment, the benchmark queries 1, 2, 3, 4 are submitted to the UI at *site1* or *site2* depending on the choice of method execution alternative (i.e. DML, DMR, or MM). These benchmark queries specify a method invocation on objects of different classes. For example, *Query 1* invokes method `Estimated_Value(...)` defined on class `Vehicle` object. A Vehicle object is assembled from six subobjects - a `VehicleTrain` object, a `PistonEngine` object, two `Company` objects, and two `Address` objects (see figure 3). Thus, the benchmark queries 1, 2, 3, 4 represent

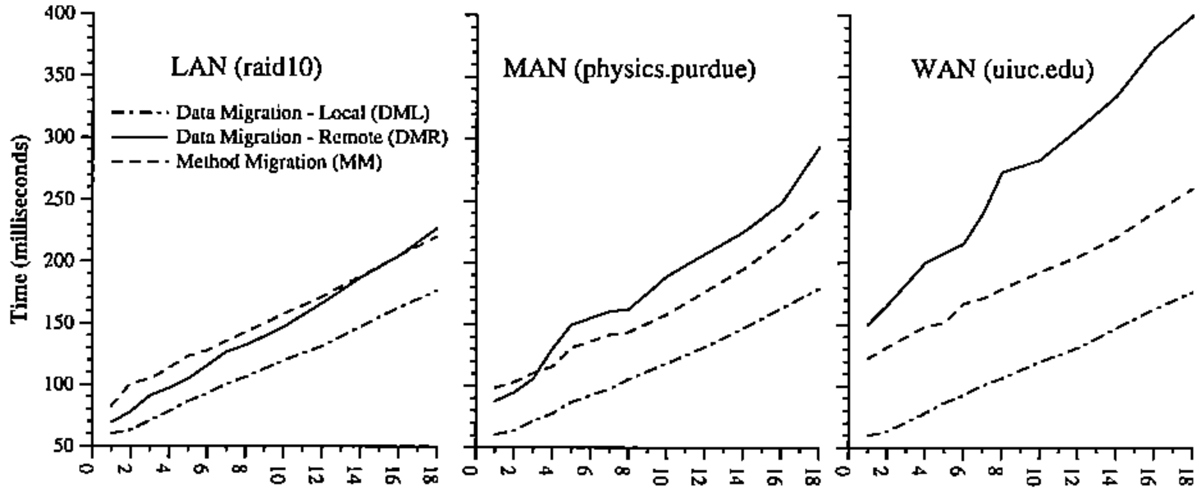

Figure 6: Total Method Execution Time v/s Size of object

Emulated Host	Comm. Delay	Data Migration - remote (DMR)					Method Migration (MM)				
		1K	2K	4K	8K	10K	1K	2K	4K	8K	10K
physics.purdue	4	88	95	130	162	189	99	103	116	143	158
ecn.purdue	6	94	111	126	170	189	98	106	122	154	167
uiuc.edu	30	150	166	208	273	283	122	132	152	179	192
pitt.edu	78	244	267	289	359	385	180	182	188	220	236
usc.edu	97	304	350	350	420	450	192	197	212	245	267

Table 1: Method Execution Time (ms) v/s Size of the Object (in WAN)

queries with methods invoked on objects requiring different number of subobjects. The cost of object assembly as well as the cost of method execution are measured.

In the second experiment, benchmark query 5 is submitted to the UI. This query invokes method on a variable length *Customer_log* object of class Text. The size of the object is varied from 1 Kilobyte to 18 Kilobytes. The actual method computation cost on the in-memory object of this class is kept independent of the object size. The intent is to determine a relationship between the the cost of method execution and the size of object, without having the need to factor out the varying method computation cost due to changing object size.

Emulated	Site1					Site2				
Host	1K	2K	4K	8K	10K	1K	2K	4K	8K	10K
LAN	60	69	86	121	137	48	55	69	96	110
physics.purdue	76	84	119	150	175	48	54	68	95	108
ecn.purdue	86	97	116	159	176	47	55	66	94	107
uiuc.edu	139	153	189	263	272	49	56	68	95	110
pitt.edu	233	254	277	342	374	48	55	69	95	112
usc.edu	293	340	339	408	438	48	59	69	96	112

Table 2: Object Assembly Cost (ms) v/s Size of the Object

5.3 Data & Discussion

Size of Object Figure 6 shows the performance of three alternatives of method execution with the varying size of object on which method is executed. The remote sites for LAN, MAN, and WAN environment are `raid10` (1 hop away), `physics.purdue` (3 hops away), and `uiuc` (18 hops away). Table 1 shows the total method execution cost for the other emulated Internet sites.

We observe that the cost of method execution on an object locally available at the query site (i.e. in DML approach) is the same in all distributed site configurations. This is expected since there is no interaction with the remote site *site1*, and therefore the invariance of method execution costs to different communication delays.

For a particular network configuration, the overheads associated with method migration – the cost of sending a method execution request to remote site and the cost of initiating a subtransaction to process this request – are relatively fixed. The difference between the cost of composing an object locally at query site via remote read requests to access data (in DMR approach) and the the cost of composing an object at remote site via local read requests to access data (in MM approach) increases with increasing object size (see Table 2). Thus, as the size of object increases, the overheads of method migration become more and more offsetted by the gains in composing the object locally at the remote site. In the LAN environment, the difference in cost of local and remote reads increases very slowly with the increase in object size. This is because the messages between the O-Raid

Benchmark Query	Method Name	Object Class	No. of Subobjects
Query 1	Estimated_Value(...)	Vehicle	7
Query 2	Imported_Engine(...)	VehicleTrain	5
Query 3	Located_In(...)	Company	2
Query 4	EducationLevel(...)	Person	1

Table 3: Subobjects required for class objects in Benchmark Queries

servers for a local read uses the same UDP/IP based communication library as is used for remote reads. Since the communication delays between two sites in LAN are low (< 1 ms), the cost of reads are similar. The method migration overheads, therefore, are counterbalanced only by gains in composing large objects locally at the remote site. On the other hand, in a WAN environment, the cost of remote reads is high (average communication delay to uiuc site is 30 ms for 64 bytes packet through the experiment). The overheads of method migration become insignificant; and the method migration approach outperforms remote data migration approach for small size objects.

Number of Subobjects As shown in figure 7, the method execution on a class object at the query site that is composed from the locally available subobjects displays the best performance in all distributed site configurations. The reasoning is same as the one for method execution on large flat objects in the previous paragraph. All the data reads for composing the in-memory object as well as the method invocation are local to the query site hence no interaction over the network.

The performance comparison of data migration from remote site and method migration to a remote site is more interesting. As in the previous paragraph, the overheads of method migration are relatively fixed for a particular sites configuration.

Since each persistent pointer stored in a class relation tuple is swizzled into an in-memory pointer by recursively fetching the tuple identified by the persistent pointer (see section ??), a complex class object with a total of "n" pointer references (direct references in the complex object or the references in the objects pointed to by the direct references, and so on) will require "n" read requests from Raid TM to compose an in-memory complex object.

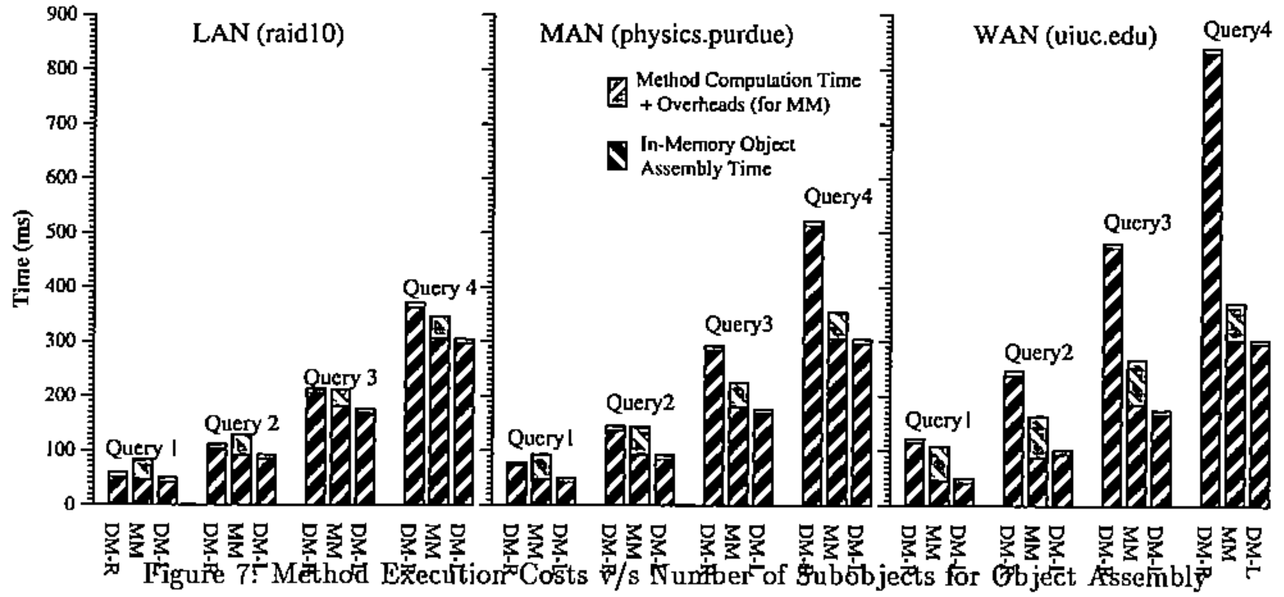

Figure 7: Method Execution Costs v/s Number of Subobjects for Object Assembly

For a particular site configuration, the difference in cost of composing a complex object at query site via remote reads to access subobject data and the cost of composing this complex object at the remote site via local reads to access subobject increases as the number of subobjects required for the complex object assembly increases.

In LAN configuration, the overheads of method migration are offsetted only for large number of remote data accesses required for composing the complex object. In our benchmark database, only *Query1* demonstrates better performance with method migration approach. This is due to the fact that local and remote access costs in LAN environment in O-Raid are very similar as explained above.

In a WAN site configuration, high communication delays to remote site (average communication delay to uiuc is 31 ms) makes remote reads expensive. Thus, the gains due to access of only a few subobjects locally at the remote site offsets the overheads of method migration; and makes the method migration approach more appealing.

Emulated Host	Comm. Delay	Data Migration - Remote (DMR)				Method Migration (MM)			
		Query1	Query2	Query3	Query4	Query1	Query2	Query3	Query4
ecn.purdue	6	510	306	145	74	364	227	143	96
pitt.edu	73	1492	824	416	210	411.7	284	199	156
usc.edu	97	2066	1106	528	260	449	316	239	182

Table 4: Total Method Execution Cost (ms) v/s Number of Subobjects referenced (in WAN)

Emulated Host	Site1				Site2			
	Query1	Query2	Query3	Query4	Query1	Query2	Query3	Query4
LAN	362	204	103	51	306	180	91	46
physics.purdue	514	284	137	72	305	181	92	46
ecn.purdue	501	298	138	66	306	180	91	47
uiuc.edu	831	473	239	115	303	186	88	45
pitt.edu	1483	817	408	202	304	176	89	44
usc.edu	2056	1098	519	252	304	178	89	45

Table 5: Object Assembly Cost (ms) v/s Number of Subobjects referenced

6 Conclusionsk

We have presented a method execution scheme called method migration, which relocates the execution of method invocation specified in the user query to a remote site where the object resides. We have implemented the support for method migration in a distributed object database called O-Raid. We have conducted an experimental study to compare the performance of method migration scheme with the traditional approach of migrating the remotely located object (data migration) to local site for method execution.

We can summarize our analysis of experimental results as follows:

1. As the cost of remote data access increases (for example, due to transition from LAN to WAN environment), the method migration approach becomes more cost effective.
2. For any network environment, as the size of remotely located object increases (for example, multimedia object), the method migration performs better than data migration approach.

Future Work We are currently working on a cost model to determine the site with optimal cost for method migration when the complex object is fragmented [7] over multiple sites. The model uses the communication latencies between the sites and the configuration of complex object on these sites to make an apriori estimate of costs of method execution at various sites.

The synchronous method migration approach presented in this paper may be too restrictive in wide area environments since it may impose long blocking periods at the query site. We are studying approaches to exploit parallelism in query execution by providing support for overlapped execution of the query at query site as well as the remote method execution site.

References

[1] B. Jenq, D. Woelk, W. Kim, and W. Lee. Query Processing in Distributed Orion. In *Int.l Conf. on Extending Database Technology, Venice*, pages 169–187, Mar 1990.

[2] E. Bertino. Query decomposition in an object-oriented database system distributed on a local area network. In *RIDE-DOM 95*, Mar 1995.

[3] Elisa Bertino and Won Kim. Indexing Techniques for Queries on Nested Objects. *IEEE Transactions on Knowledge and Data Engineering*, 1(2), June 1989.

[4] P. Valduriez. Join Indices . *ACM Transactions on Database Systems*, 12(2), Jun 1987.

[5] Elisa Bertino. Method precomputation in object-oriented databases. In *Proceedings of Organizational Computing Systems, Atlanta, Georgia*, Nov 1991.

[6] A. Kemper, C. Kilger, and G. Moerkette. Function Materialization in Object Bases. In *ACM SIGMOD*, pages 258–267, 1991.

[7] Jagannathan Srinivasan. *Replication and Fragmentation of Composite Objects in Distributed Database Systems.* PhD thesis, Purdue University, Aug 1992.

[8] B. Bhargava, Y. Jiang, and J. Srinivasan. O-Raid: Experiences and Experiments. In *Proceedings of the Int.l Conf. on Intelligent and Cooperative Information Systems*, May 1993.

[9] Bharat Bhargava and John Riedl. The RAID Distributed Database System. *IEEE Transactions on Software Engineering*, 16(6), June 1989.

[10] S. Ford, J. Blakeley, and T. Bannon. Open OODB: A Modular Object-Oriented DBMS. In *ACM Sigmod*, pages 552–553, Washington, DC, May 1993.

[11] Won Kim. UNISQL/X: Unified Relational and Object-Oriented Database System. *Sigmod Record*, 23(2), 1994.

[12] Nicole Melander. Personal communications. In *Oracle Corporation, Bethesda, Maryland, USA*, February 1995.

[13] Yongguang Zhang and Bharat Bhargava. WANCE: A Wide Area Network Communication Emulation System. In *Proceedings of the IEEE Workshop on Advances in Parallel and Distributed Systems*, pages 40–45, Princeton, New Jersey, October 1993.

[14] B. Stroustrup. *The C++ Programming Language*. Addison-Wesley, Reading, Mass., 1986.

[15] J. G. Mullen, J. Srinivasan, P. Dewan, and B. Bhargava. Supporting Queries in the O-Raid Object-Oriented Database System. In *Proceedings of the International Computer Software and Applications Conference*, 1990.

[16] P. Dewan, A. Vikram, and B. Bhargava. Engineering the Object-Relation Model in O-Raid. In *Proceedings Of the International Conference on Foundations of Data Organization and Algorithms*, pages 389–403, June 1989.

[17] Wilson W. Ho and Ronald A. Olsson. An Approach to Genuine Dynamic Linking. *Software - Practice and Experience*, 21(4):375–390, april 1991.

[18] Yongguang Zhang and Bharat Bhargava. A Study of Distributed Transaction Processing in Wide Area Networks. Technical Report CSD-TR-94-016, Department of Computer Sciences, Purdue University, January 1994.